Flickr8K dataset:

Flickr8K_Dataset: https://drive.google.com/file/d/1u3oqx36XApnAykFDB6EEWUIfd_CxRQQ9/view
(Contains around 8K images (8091))
Flickr8K_text: https://drive.google.com/file/d/1qcRy3WpQv4dGtu65gETtYLWxDPBrRtx1/view
(Contains captions

1) Flickr8k.lemma.token.txt : 5 captions per image for all 8K images.
2) Flickr_8k.trainImages.txt : List of 6K training images set.
3) Flickr_8k.testImages.txt : List of 1K test images set.
4) Flickr_8k.devImages.txt : List of 1K dev/validation images set.

)

In `main.py`:

Data Preprocessing and Cleaning

1) **Load caption data** using `load_doc()` by reading the text file as a single string.
2) **Parse raw captions** using `all_img_captions()`, grouping multiple captions per image into a dictionary.
3) **Clean caption** with `cleaning_text()` by removing punctuation, converting to lowercase, filtering out short and non-alphabetic words.
4) **Build a vocabulary** of unique words from the cleaned captions using `text_vocabulary()`.
5) **Save processed captions** to a new file (`descriptions.txt`) in a structured format.

We have cleaned 5 captions per image saved in "**descriptions.txt"** for all 8K images.

Image Feature Extraction Using Pretrained CNN (Xception)

6) **Download pretrained Xception weights** from TensorFlow's model zoo.**Initializes the Xception model** (excluding top classification layer) for feature extraction using `include_top=False`.
7) **Initialize the Xception model** (excluding top classification layer) for feature extraction using `include_top=False`.
8) **Define `extract_features()`** to process each image:
   Loads and resizes the image to 299×299 (Xception input size),
   Normalizes pixel values,
   Passes it through the Xception model to extract a **2048-dimensional feature vector**.
9) **Save extracted image features** as a dictionary using `pickle` (`features.p`) for future use.

We have extracted features from all 8K images and saved it in **"features.p"**

Preparing Training Data

Let's start working with training dataset:

10) **`load_photos()`**:Load a list of training image filenames from a text file and filter only those images that exist in the training dataset directory.
11) **`load_clean_descriptions()`**: Load preprocessed (cleaned) image captions from a file, keeping only those that correspond to the training images, and wraps each caption with `<start>` and `<end>` tokens.
12) **`load_features()`**: Load the saved image features (`features.p`) and filter them to include only features corresponding to the training images.

Tokenizer

Let's make tokenizer:

13) **`dict_to_list()`**: Convert the dictionary of image-to-captions into a flat list of all captions.**`create_tokenizer()`**: Use Keras's `Tokenizer` to fit on the list of captions and map words to unique integer indices.
14) **`create_tokenizer()`**: Use Keras's `Tokenizer` to fit on the list of captions and map words to unique integer indices.
15) **Save the tokenizer** using `pickle` for future use during training and inference.

Conversion of captions into training sequences

16) Calculate **vocab_size**: the total number of unique words in the dataset and ompute the maximum length of all training captions.
17) **`create_sequences()`**

    For each caption:

    Converts it to a sequence of integers using the tokenizer.
    Generates multiple input-output pairs where:
    Input = image feature + partial caption (padded),
    Output = next word (one-hot encoded).

- Example: `'a dog runs'` →

    Input: `['a']` → Output: `'dog'`

    Input: `['a', 'dog']` → Output: `'runs'`

18) **`data_generator()`**

    Creates a `tf.data.Dataset` generator that:

    Yields batches of `{image feature, partial caption} → next word` pairs,
    Matches the model's expected input-output shape using `tf.TensorSpec`.

Model Architecture and Training

19) Model Definition:

    Image input branch:

    Accepts 2048-dimensional image feature vectors (from CNN),
    Applies dropout and reduces to 256 features using a dense layer.

    Caption input branch:

    Accepts tokenized caption sequences (max length),
    Embeds words into 256-dimensional vectors,
    Processes the sequence using an LSTM to capture context.

    Merging branches:

    Combines image features and LSTM output using `add()`,
    Passes through dense layers to predict the next word using a `softmax` layer.

    Compiles the model with `categorical_crossentropy` loss and `adam` optimizer.

20) Model training:

    Runs training for 10 epochs, After each epoch, saves the model to a new `.h5` file inside the `models/` directory.

In `evaluate.py`:

1) **Defines helper functions** to:
   Load image filenames, captions, and pre-extracted image features,
   Convert predicted word indices back to words,
   Generate captions for images using the trained model.
2) **Defines a multimodal model** architecture that combines CNN image features and LSTM-generated text context, to predict the next word in the caption sequence.
3) **Loads pre-trained tokenizer and sets vocab size and max caption length**.
4) **Loads test image features and cleaned test captions**.
5) **Loads model weights from a `.h5` file**, compiles the model, and prepares it for evaluation.
6) **Generates captions for test images** and compares them to ground-truth captions using **BLEU scores (1–4)**.
7) **Prints BLEU scores** as a quantitative evaluation of the model's captioning accuracy.

Results:

The model was trained with a vocabulary size of **7,577** and a maximum caption length of **32**. It contained 5,002,649 trainable parameters. Training was conducted for 10 epochs, with the loss steadily decreasing from **5.03 to 2.92**, indicating good convergence.

7700/7700 ━━━━━━━━━━━━━━━━━━━━ 282s 37ms/step - loss: 5.0365
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 266s 34ms/step - loss: 3.8218
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 278s 36ms/step - loss: 3.4940
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 276s 36ms/step - loss: 3.3104
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 279s 36ms/step - loss: 3.1918
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 280s 36ms/step - loss: 3.1070
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 280s 36ms/step - loss: 3.0502
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 281s 36ms/step - loss: 2.9944
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 283s 37ms/step - loss: 2.9602
7700/7700 ━━━━━━━━━━━━━━━━━━━━ 280s 36ms/step - loss: 2.9220

BLEU Score of validation set for 10 epochs, demonstrate general improvement and then stabilization.
Epoch 0: BLEU-1=0.3624, BLEU-2=0.1912, BLEU-3=0.1199, BLEU-4=0.0487
Epoch 1: BLEU-1=0.3458, BLEU-2=0.1856, BLEU-3=0.1206, BLEU-4=0.0489
Epoch 2: BLEU-1=0.2871, BLEU-2=0.1571, BLEU-3=0.1060, BLEU-4=0.0415
Epoch 3: BLEU-1=0.3150, BLEU-2=0.1680, BLEU-3=0.1095, BLEU-4=0.0422
Epoch 4: BLEU-1=0.3108, BLEU-2=0.1649, BLEU-3=0.1067, BLEU-4=0.0408
Epoch 5: BLEU-1=0.3431, BLEU-2=0.1860, BLEU-3=0.1265, BLEU-4=0.0539
Epoch 6: BLEU-1=0.3546, BLEU-2=0.1904, BLEU-3=0.1281, BLEU-4=0.0549
Epoch 7: BLEU-1=0.3345, BLEU-2=0.1798, BLEU-3=0.1174, BLEU-4=0.0452
Epoch 8: BLEU-1=0.3589, BLEU-2=0.1953, BLEU-3=0.1332, BLEU-4=0.0574
Epoch 9: BLEU-1=0.3451, BLEU-2=0.1871, BLEU-3=0.1269, BLEU-4=0.0535

During validation, the best BLEU-4 score was achieved at epoch 8, with BLEU-4=**0.0574**
On the test set, the model from the last epoch (epoch 9) achieved:
BLEU-1: 0.333130 BLEU-2: 0.181531 BLEU-3: 0.120728 BLEU-4: 0.047832

However, the model from epoch 8 (with best validation BLEU-4) gave improved results on the test set:
BLEU-1: 0.346744 BLEU-2: 0.185889 BLEU-3: 0.122901 BLEU-4: 0.051291

This project successfully demonstrates image captioning by integrating pretrained CNN features with LSTM-based language modeling on the Flickr8k dataset. The model achieved a best BLEU-4 score of **0.0574 on validation** and **0.0513 on test data**